Software Engineering, Formal Methods, and Computational Thinking

FUTATSUGI, Kokichi
（二木 厚吉）
JAIST
(Japan Advanced Institute of Science and Technology)
What I am going to talk today

What is software?
-- What is its nature? --

What is software engineering?

What are formal methods?

What is computational thinking?

What is to be competent in software?

What are ACM code of ethics and professional conduct?
What is software?

Documents that relate to computer programs: Requirement Spec, Design Spec, Source program codes, Executable programs dose, Reference manuals, Operation manuals, Verification documents, etc.

from the point of production/development
Software is discrete

Principles in Physics and Chemistry are described with differential equations. (continuous systems)

Software is documents, i.e. sequences of symbols, and their properties are described with formal language theory, symbolic logics, and algorithm theory. (discrete systems)

Can software’s semantics be described with symbolic logics? What is semantics?
Software should cope with a huge number

Run-time of a computer

Total number of cases:

$2^2$ time

$2^{100}$ $2^{10000}$

One case determines lift or death

(discrete systems)
Software is open and liberal  
- Software promotes openness -

Software is digital, global, and universal

<table>
<thead>
<tr>
<th>Railways</th>
<th>↔</th>
<th>Roads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telephone network</td>
<td>↔</td>
<td>Internet</td>
</tr>
<tr>
<td>Guaranteed Systems</td>
<td>↔</td>
<td>Best Effort Systems</td>
</tr>
<tr>
<td>Functional Properties</td>
<td>↔</td>
<td>Non-Functional Properties</td>
</tr>
<tr>
<td>Technical Systems</td>
<td>↔</td>
<td>Socio-Technical Systems</td>
</tr>
<tr>
<td>Closed Systems</td>
<td>↔</td>
<td>Open Systems</td>
</tr>
</tbody>
</table>
What is software engineering?

Science/technology/engineering for development/operation/disengagement of software
Text books on software engineering (1)

- 1st Edition 1982
- 2nd Edition 1984
- 3rd Edition 1989
- 5th Edition 1995
- 7th Edition 2004
- 9th Edition 2010
- 10th Edition 2015
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>17. Rapid software development</td>
</tr>
<tr>
<td>2. Socio-technical systems</td>
<td>18. Software reuse</td>
</tr>
<tr>
<td>4. Software processes</td>
<td>20. Critical systems development</td>
</tr>
<tr>
<td>5. Project management</td>
<td>21. Software evolution</td>
</tr>
<tr>
<td>6. Software requirements</td>
<td>22. Verification and validation</td>
</tr>
<tr>
<td>7. Requirements engineering process</td>
<td>23. Software testing</td>
</tr>
<tr>
<td>9. Critical systems specification</td>
<td>25. People management</td>
</tr>
<tr>
<td>11. Software architecture</td>
<td>27. Quality management</td>
</tr>
<tr>
<td>12. Distributed systems architecture</td>
<td>28. Process improvement</td>
</tr>
<tr>
<td>13. Application architectures</td>
<td>29. Configuration management</td>
</tr>
<tr>
<td>15. Real-time design</td>
<td>31: Service-oriented software engineering</td>
</tr>
</tbody>
</table>
Contents of Sommerville’s text book (ver9)(2010)
http://www.cs.st-andrews.ac.uk/~ifs/Books/SE9/

1 Introduction
2 Software Processes
3 Agile software development
4 Requirements engineering
5 System modeling
6 Architectural design
7 Design and Implementation
8 Software testing
9 Software Evolution
10 Socio-technical Systems
11 Dependability and Security
12 Dependability and Security Specification
13 Dependability Engineering
14 Security Engineering
15 Dependability and Security Assurance
16 Software Reuse
17 Component-based Software Engineering
18 Distributed Software Engineering
19 Service-oriented Architecture
20 Embedded Systems
21 Aspect-oriented software engineering
22 Project management
23 Project planning
24 Quality management
25 Configuration management
26 Process improvement

[Web Chapters]
27 Formal Specification
28 Application Architectures
29 Interaction Design
30 Documentation
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Software Processes</td>
</tr>
<tr>
<td>3</td>
<td>Agile Software Development</td>
</tr>
<tr>
<td>4</td>
<td>Requirements Engineering</td>
</tr>
<tr>
<td>5</td>
<td>System Modeling</td>
</tr>
<tr>
<td>6</td>
<td>Architectural Design</td>
</tr>
<tr>
<td>7</td>
<td>Design and Implementation</td>
</tr>
<tr>
<td>8</td>
<td>Software Testing</td>
</tr>
<tr>
<td>9</td>
<td>Software Evolution</td>
</tr>
<tr>
<td>10</td>
<td>Dependable Systems</td>
</tr>
<tr>
<td>11</td>
<td>Reliability Engineering</td>
</tr>
<tr>
<td>12</td>
<td>Safety Engineering</td>
</tr>
<tr>
<td>13</td>
<td>Security Engineering</td>
</tr>
<tr>
<td>14</td>
<td>Resilience Engineering</td>
</tr>
<tr>
<td>15</td>
<td>Software Reuse</td>
</tr>
<tr>
<td>16</td>
<td>Component-based Software Engineering</td>
</tr>
<tr>
<td>17</td>
<td>Distributed Software Engineering</td>
</tr>
<tr>
<td>18</td>
<td>Service-oriented Software Engineering</td>
</tr>
<tr>
<td>19</td>
<td>Systems Engineering</td>
</tr>
<tr>
<td>20</td>
<td>Systems of Systems</td>
</tr>
<tr>
<td>21</td>
<td>Real-time Software Engineering</td>
</tr>
<tr>
<td>22</td>
<td>Project management</td>
</tr>
<tr>
<td>23</td>
<td>Project planning</td>
</tr>
<tr>
<td>24</td>
<td>Quality management</td>
</tr>
<tr>
<td>25</td>
<td>Configuration management</td>
</tr>
<tr>
<td>[web chapters]</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Process Improvement</td>
</tr>
<tr>
<td>27</td>
<td>Formal Specification</td>
</tr>
<tr>
<td>28</td>
<td>Application Architectures</td>
</tr>
<tr>
<td>29</td>
<td>Interaction Design</td>
</tr>
<tr>
<td>30</td>
<td>Documentation</td>
</tr>
<tr>
<td>31</td>
<td>Aspect-oriented Software Engineering</td>
</tr>
</tbody>
</table>
Software engineering amounts to be generic “systems engineering”

Programming/coding gradually forms an independent technical area; SE is more than that

Modeling/analysis/verification technologies in problem domains have become important foundations of software engineering

Components and structure are still important topics ➔ Software Architecture, OO modeling, Multi-Agent

Importance of non-functional properties (Evolution, reliability, safety, security, mobility) is increasing
Text books on software engineering (2)
Dines Bjorner’s text books (2006)
www2.imm.dtu.dk/~db/Software_Engineering/

Contents of Three Volumes

[1] **Abstraction and Modelling**
- xxxix + 711 Pages

[2] **Specification of Systems and Languages**
- xxiv + 779 Pages

- xxx + 766 Pages

Prof. Dines Bjorner has stayed at KF’s lab in JAIST from 2006.2 to 2007.1, and did the research on “Formal Methods and Domain Engineering”
Software engineering, domain engineering, requirement engineering

The name software engineering was given for expecting an engineering discipline based on solid theory and precise calculus like other engineering (1968, NATO Science Committee)

Other engineering like mechanical engineering or chemical engineering handles an already established problem domain. Software engineering is, however, expected to handle any problem domain, and modeling/analyses/formalization of domains are major tasks of software engineering.
→ domain engineering, requirement engineering

Domain engineering and requirement engineering based on forms and calculus are becoming important topics in software engineering.
Characteristics of software engineering

♦ Becoming generic engineering discipline for development/operation/disengagement of systems
  • Application domains are expanding
    (software makes every dream come true)
♦ Engineering discipline for description and implementation of systems based of computing machinery
♦ The most important basic technology for 21st century where the world wide computer network is the most important social infra-structure
Basic facts on computer and software

A computer is manipulating “forms”
(not “semantics”)

Software engineering (or formal method) is an
discipline for studying model/theory/technique/
method to translate problem domain (i.e. semantics)
into machine (i.e. forms).
What are formal methods?

form without content $\leftrightarrow$ form representing essence

technology $\leftrightarrow$ science

practice $\leftrightarrow$ theory
Formal methods

♦ Software development methods based on mathematical logics and discrete math.
  • Specification/program/verification are constructed based on math models like set, function, algebra

♦ Software development methods based on formal specifications
  • Verification can be done based on formal specifications

The highest level technology for developing highly reliable systems (e.g. ITSEC)
Problems, Specifications, Machines

Requirements
Environment Phenomena

Application Env./Aria

Spec interface
Specification

by Michael Jackson
Specification is better to be refutable, consistent, complete, unambiguous, and transparent

♦ Refutable: true or false can be decided objectively
♦ Consistent: no contradicting statements
♦ Complete: all facts are stated
♦ Unambiguous: no way to interpret in multiple ways
♦ Transparent: easy to understand
Informal Specification

♦ Described in natural languages with diagrams and pictures; there is no rules to determine the meaning of the description objectively
♦ Meaning can not be determined objectively, and not refutable
♦ It is difficult to judge whether a description is consistent, complete, or ambiguous
♦ Usable in practice, but it is important to know the limits
Formal Specification

- There are predefined rules for determining the meaning of the specifications.
- It is highly possible to determine whether the specification is refutable, consistent, complete, unambiguous.
- Written in formal languages and can be processed by computers.
- It is necessary to study the language for writing formal specs, but once studied, it is easy to write and read them objectively.
Formal specification languages and programming languages

♦ Formal specification language
  = modeling language for human beings
♦ Programming language
  = language for instructing computers
♦ Formal specification language is ultra high level programming language
  • Functional programming languages
  • Logic programming languages
  • Algebraic programming languages
Primary use of formal specifications

♦ formalization of concepts in a problem domain, and define vocabulary for the domain
  • Vocabulary: names for collections of objects + names for operations on them
emedOntology
emedMichael Jackson’s designation
emedSignature and module structure

It is non-sense to use precise method if a problem is not clear. (John von Neumann)

formal == rigorous and precise
Contrasting uses of formal methods (1)

♦ Communication tool (formal spec languages)
  • Requirement specs, design specs, …
    ♥ Documentation in formal languages
  • International standards for system interfaces
    ♥ formal documentation ➔ scientific maintenance

♦ Verification tool (theorem proving, model checking)
  • For highly reliable systems
    ♥ Aviation, railway, automobile, hospital …
      o No other way ➔ ITSEC standard
Contrasting uses of formal methods (2)

♦ Modeling/spec/verification of problems/systems (pre-coding or without-coding)
  interactive verification with formal spec language system
  Specification and verification of
  • requirement and domain
  • security policy and business rules
  • Social systems
  • Bio-systems

♦ Modeling/spec/verification of computer programs (post-coding)
  static analyses and semi-automatic verification
  • Verifying compiler (for million of lines program)
    ❤ Verified compilers, verified operating systems
  • Application generators generating verified codes
    ❤ verified web services generators
CafeOBJ is a most advanced formal specification language which inherits many advanced features (e.g. flexible mix-fix syntax, powerful and clear typing system with ordered sorts, parameteric modules and views for instantiating the parameters, and module expressions, etc.) from OBJ (or more exactly OBJ3) algebraic specification language.
CafeOBJ (2)  
https://cafeobj.org

CafeOBJ is a language for writing formal (i.e. mathematical) specifications of models for wide varieties of software and systems, and verifying properties of them. CafeOBJ implements equational logic by rewriting and can be used as a powerful interactive theorem proving system. Specifiers can write proof scores also in CafeOBJ and doing proofs by executing the proof scores.
CafeOBJ has state-of-art rigorous logical semantics based on institutions. The CafeOBJ cube shows the structure of the various logics underlying the combination of the various paradigms implemented by the language. Proof scores in CafeOBJ are also based on institution based rigorous semantics, and can be constructed using a complete set of proof rules.
J. Wing’s Computational Thinking

https://www.cs.cmu.edu/~15110-s13/Wing06-ct.pdf

A universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.
The characteristics that define computational thinking are decomposition, pattern recognition / data representation, generalization/abstraction, and algorithms.

By decomposing a problem, identifying the variables involved using data representation, and creating algorithms, a generic solution results.

The generic solution is a generalization or abstraction that can be used to solve a multitude of variations of the initial problem.

https://en.wikipedia.org/wiki/Computational_thinking
The phrase computational thinking was brought to the forefront of the computer science community as a result of an ACM Communications article on the subject by Jeannette Wing.

The article suggested that thinking computationally was a fundamental skill for everyone, not just computer scientists, and argued for the importance of integrating computational ideas into other disciplines.
Characteristics (1)
https://en.wikipedia.org/wiki/Computational_thinking

Computational Thinking is a problem-solving process that includes (but is not limited to) the following characteristics:

- Formulating problems in a way that enables us to use a computer and other tools to help solve them;

- Logically organizing and analyzing data;

- Representing data through abstractions such as models and simulations;
Characteristics (2)

https://en.wikipedia.org/wiki/Computational_thinking

- Automating solutions through algorithmic thinking (a series of ordered steps);

- Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources;

- Generalizing and transferring this problem solving process to a wide variety of problems.
Computational thinking builds on the power and limits of computing processes, whether they are executed by a human or by a machine.

Computational methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone.

Computational thinking is a fundamental skill for everyone, not just for computer scientists.
Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science.

Computational thinking is reformulating a seemingly difficult problem into one we know how to solve, perhaps by reduction, embedding, transformation, or simulation.

Computational thinking is thinking recursively.
Computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system.

(CT is) Conceptualizing, not programming.

(CT suggests) Computer science is not computer programming.

(CT suggests) Thinking like a computer scientist means more than being able to program a computer. It requires thinking at multiple levels of abstraction.

Quotations (3) from https://www.cs.cmu.edu/~15110-s13/Wing06-ct.pdf
Quotations (4) from https://www.cs.cmu.edu/~15110-s13/Wing06-ct.pdf

- (CT is) Fundamental, not rote skill.
- (CT is) A way that humans, not computers, think.
- (CT) Complements and combines mathematical and engineering thinking.
- (CT is) Ideas, not artifacts.
- (CT is) For everyone, everywhere.
- (CT suggests) One can major in computer science and do anything.
What is to be competent in software?
abstract world <-> concrete world

abstract world

230×3=690

3 bulbs of 230 Yen each

concrete world

690 Yen
competent in programming ≠
competent in software

competent in programming
(competent in calculus)

competent in software
(competent in math)
Commitment to ethical professional conduct is expected of every member (voting members, associate members, and student members) of the Association for Computing Machinery (ACM).
1. GENERAL MORAL IMPERATIVES

As an ACM member I will ....

1.1 Contribute to society and human well-being.
1.2 Avoid harm to others.
1.3 Be honest and trustworthy.
1.4 Be fair and take action not to discriminate.
1.5 Honor property rights including copyrights and patent.
1.6 Give proper credit for intellectual property.
1.7 Respect the privacy of others.
1.8 Honor confidentiality.
2. MORE SPECIFIC PROFESSIONAL RESPONSIBILITIES

As an ACM computing professional I will ....

2.1 Strive to achieve the highest quality, effectiveness and dignity in both the process and products of professional work.

2.2 Acquire and maintain professional competence.

2.3 Know and respect existing laws pertaining to professional work.

2.4 Accept and provide appropriate professional review.

2.5 Give comprehensive and thorough evaluations of computer systems and their impacts, including analysis of possible risks.

2.6 Honor contracts, agreements, and assigned responsibilities.

2.7 Improve public understanding of computing and its consequences.

2.8 Access computing and communication resources only when authorized to do so.
3. ORGANIZATIONAL LEADERSHIP IMPERATIVES

As an ACM member and an organizational leader, I will ....

3.1 Articulate social responsibilities of members of an organizational unit and encourage full acceptance of those responsibilities.

3.2 Manage personnel and resources to design and build information systems that enhance the quality of working life.

3.3 Acknowledge and support proper and authorized uses of an organization's computing and communication resources.

3.4 Ensure that users and those who will be affected by a system have their needs clearly articulated during the assessment and design of requirements; later the system must be validated to meet requirements.

3.5 Articulate and support policies that protect the dignity of users and others affected by a computing system.

3.6 Create opportunities for members of the organization to learn the principles and limitations of computer systems.
4. COMPLIANCE WITH THE CODE

As an ACM member I will ....

4.1 Uphold and promote the principles of this Code.
4.2 Treat violations of this code as inconsistent with membership in the ACM.